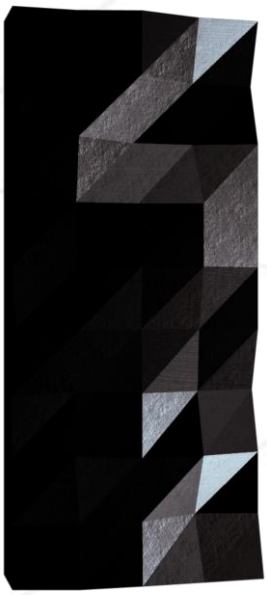

C++ - Módulo 01

Alocação de memória, apontadores para os membros, referências, declaração de mudança

Resumo:
Este documento contém os exercícios do Módulo 01 dos módulos C++.

Versão: 9

Conteúdos

Capítulo I

Introdução

C++ é uma linguagem de programação de uso geral criada por Bjarne Stroustrup como uma ex- tensão da linguagem de programação C, ou "C com Classes" (fonte: Wikipedia).

O objectivo destes módulos é apresentar-lhe a **Programação Orientada para Objectos**. Este será o ponto de partida da sua viagem em C++. Muitas línguas são recomendadas para aprender o OOP. Decidimos escolher C++ porque deriva do seu velho amigo C. Como se trata de uma linguagem complexa, e para manter as coisas simples, o seu código respeitará a norma C++98.

Estamos conscientes de que o C++ moderno é muito diferente em muitos aspectos. Portanto, se quiser tornar-se um desenvolvedor C++ proficiente, cabe-lhe a si ir mais longe depois do 42 Common Core!

Capítulo II

Regras gerais

Compilação

- Compilar o seu código com c++ e as bandeiras -Wall -Wextra -Werror
- O seu código ainda deve ser compilado se adicionar a bandeira -std=c++98

Formatação e convenções de nomeação

- Os directórios de exercícios serão nomeados desta forma: ex00, ex01, ... , exn
- Nomeie os seus ficheiros, classes, funções, funções de membro e atributos, conforme exigido nas directrizes.
- Escrever os nomes das classes no formato **UpperCamelCase.** Os ficheiros contendo o código da classe serão sempre nomeados de acordo com o nome da classe. Por exemplo: ClassName.hpp/ClassName.h, ClassName.cpp, ou ClassName.tpp. Então, se tiver um ficheiro de cabeçalho contendo a definição de uma classe "BrickWall" que representa uma parede de tijolo, o seu nome será BrickWall.hpp.
- Salvo especificação em contrário, todas as mensagens de saída devem ser terminadas por um novo carácter de linha e apresentadas na saída padrão.
- *Adeus Norminette!* Nenhum estilo de codificação é aplicado nos módulos C++. Pode seguir o seu favorito. Mas tenha em mente que um código que os seus avaliadores não conseguem compreender é um código que eles não conseguem avaliar. Faça o seu melhor para escrever um código limpo e legível.

Permitido/Proibido

Já não está a codificar em C. Tempo para C++! Portanto:

- É-lhe permitido utilizar quase tudo da biblioteca padrão. Assim, em vez de se cingir ao que já sabe, seria inteligente utilizar o máximo possível as versões C++-ish das funções C a que está habituado.
- No entanto, não se pode utilizar qualquer outra biblioteca externa. Isto significa que as bibliotecas C++11 (e formas derivadas) e Boost são proibidas. As seguintes funções são também proibidas: *printf(), *alloc() e free(). Se as usar, a sua nota será 0 e ponto final.

- Note que, salvo indicação explícita em contrário, a utilização do namespace <ns_name> e
  são proibidas as palavras-chave de amigos. Caso contrário, a sua nota será -42.
- **É-lhe permitido utilizar o STL apenas no Módulo 08.** Isto significa: sem **Containers** (vector/lista/mapa/e assim por diante) e sem **Algoritmos** (qualquer coisa que requeira incluir o <algoritmo> cabeçalho) até lá. Caso contrário, a sua nota será -42.

Alguns requisitos de concepção

- O vazamento de memória também ocorre em C++. Quando se atribui memória (utilizando o novo
  palavra-chave), deve evitar **fugas de memória**.
- Do Módulo 02 ao Módulo 08, as suas aulas devem ser concebidas na **Forma Canónica Ortodoxa, excepto quando explícitamente indicado em contrário.**
- Qualquer implementação de função colocada num ficheiro de cabeçalho (excepto para os modelos de funções) significa 0 para o exercício.
- Deverá poder utilizar cada um dos seus cabeçalhos independentemente dos outros. Assim, devem incluir todas as dependências de que necessitam. No entanto, deve evitar o problema da dupla inclusão, acrescentando **guardas de inclusão**. Caso contrário, a sua classificação será 0.

Leia-me

- Pode adicionar alguns ficheiros adicionais se precisar (ou seja, para dividir o seu código). Como estas atribuições não são verificadas por um programa, sinta-se à vontade para o fazer, desde que entregue os ficheiros obrigatórios.
- Por vezes, as directrizes de um exercício parecem curtas, mas os exemplos podem mostrar requisitos que não estão explicitamente escritos nas instruções.
- Leia cada módulo completamente antes de começar! A sério, fá-lo.
- Por Odin, por Thor! Usa o teu cérebro!!!

Terá de implementar muitas aulas. Isto pode parecer aborrecido, a menos que seja capaz de escrever o seu editor de texto favorito.

É-lhe dada uma certa liberdade para completar os exercícios. No entanto, siga as regras obrigatórias e não seja preguiçoso. Iria perder muita informação útil! Não hesite em ler sobre conceitos teóricos.

Capítulo III

Exercício 00: BraiiiiiiinnnnnzzzZ

	Exercício : 00
BraiiiiiiinnnnnzzzZ	
Anuário de entrada : *ex00/*	
Ficheiros para entregar : Makefile, main.cpp, Zombie.{h, hpp}, Zombie.cpp, newZombie.cpp, randomChump.cpp	
Funções proibidas : Nenhuma	

Primeiro, implementar uma classe **Zombie.** Tem um nome de atributo privado de cadeia.
Acrescentar um anúncio de função de membro nulo (nulo **);** à classe Zombie . Os zombies anunciam-se como se segue:

<nome>: BraiiiiiiinnnnnzzzZ...

Não imprimir os parênteses dos ângulos (< e >). Para um zombie chamado Foo, a mensagem seria:

Foo: BraiiiiiiinnnnnzzzZ...

Em seguida, implementar as duas funções seguintes:

- Zombie* newZombie(std::string name **);**
  Cria um zombie, nomeia-o, e devolve-o para que possa utilizá-lo fora do âmbito da função.
- void randomChump(std::string name);
  Cria um zombie, nomeia-o, e o zombie anuncia-se a si próprio.

Agora, qual é o verdadeiro objectivo do exercício? É preciso determinar em que caso é melhor alocar os zombies na pilha ou pilha.

Os zombies devem ser destruídos quando já não precisam deles. O destruidor deve imprimir uma mensagem com o nome do zombie para efeitos de depuração.

Capítulo IV

Exercício 01: Moar brainz!

	Exercício : 01
Moar brainz!	
Anuário de entrada : *ex01/*	
Ficheiros para entregar : Makefile, main.cpp, Zombie.{h, hpp}, Zombie.cpp, ZombieHorde.cpp	
Funções proibidas : Nenhuma	

Hora de criar uma **horda de zombies**!

Implementar a seguinte função no ficheiro apropriado:

```
Zumbi*       zombieHorde( int N, std::string name );
```

Deve atribuir N objectos Zombie numa única atribuição. Depois, tem de inicializar os zombies, atribuindo a cada um deles o nome passado como parâmetro. A função devolve um ponteiro ao primeiro zombie.

Implemente os seus próprios testes para assegurar que a sua função zombieHorde() funciona como esperado.
Tente anunciar() para cada um dos zombies.

Não se esqueça de apagar todos os zombies e verificar se há **fugas de memória**.

Capítulo V

Exercício 02: O QUE É ESTE CRÉDITO

	Exercício : 02
OLÁ ESTE É O CÉREBRO	
Directório de entrada :	
ex02/	
Ficheiros para entregar : Makefile, main.cpp	
Funções proibidas : Nenhuma	

Escreva um programa que contenha:

- Uma variável de corda inicializada para "HI THI THIS BRAIN".
- stringPTR: Um ponteiro para a corda.
- stringREF: Uma referência à corda.

O seu programa tem de ser impresso:

- O endereço de memória da variável string.
- O endereço de memória mantido por stringPTR.
- O endereço de memória mantido por

stringREF. E depois:

- O valor da variável string.
- O valor apontado por stringPTR.
- O valor apontado por stringREF.

É tudo, sem truques. O objectivo deste exercício é desmistificar referências que podem parecer completamente novas. Embora existam algumas pequenas diferenças, esta é outra sintaxe para algo que já se faz: abordar a manipulação.

Capítulo VI

Exercício 03: Violência desnecessária

	Exercício : 03
Violência desnecessária	
Directório de entrada : *ex03/*	
Ficheiros para entregar : Makefile, main.cpp, Weapon.{h, hpp}, Weapon.cpp, HumanA.{h, hpp}, HumanA.cpp, HumanB.{h, hpp}, HumanB.cpp	
Funções proibidas : Nenhuma	

Implementar uma classe de Armas que tenha:

- Um tipo de atributo privado, que é uma corda.
- Uma função de membro getType() que devolve uma referência constante ao tipo.
- Uma função de membro setType() que define o tipo usando a nova função passada como paraméter.

Agora, crie duas classes: **HumanA** e **HumanB**. Ambas têm uma Arma e um nome. Têm também um ataque de função de membro() que exibe (claro, sem os parênteses angulares):

<nome> ataques com o seu <tipo de arma>

HumanA e HumanB são quase os mesmos, excepto por estes dois pequenos detalhes:

- Enquanto a HumanA toma a Arma no seu construtor, a HumanB não o faz.
- HumanB pode **não** ter **sempre** uma Arma, enquanto que HumanA estará **sempre** armada.

Se a sua implementação estiver correcta, a execução do seguinte código imprimirá um ataque com "taco de picos brutos", depois um segundo ataque com "algum outro tipo de taco" para ambos os casos de teste:

```
int main()
{
  {
    Clube de armas = Weapon("clube de espigão bruto");

    HumanB jim("Jim"); jim.
    setWeapon(club); jim.attack();
    setType("algum outro tipo de clube"); jim.attack();
  }

  devolver 0;
}
```

Em que caso acha que seria melhor usar um ponteiro para Arma? E uma referência a Weapon? Porquê? Pense nisso antes de iniciar este exercício.

Capítulo VII

Exercício 04: Sed é para os perdedores

	Exercício : 04
Sed é para os perdedores	
Anuário de entrada : *ex04/*	
Ficheiros para entregar : Makefile, main.cpp, *.cpp, *.{h, hpp}	
Funções proibidas : std::string::substituir	

Crie um programa que tome três parâmetros na seguinte ordem: um nome de ficheiro e duas cordas, s1 e s2.

Irá abrir o ficheiro <filename> e copia o seu conteúdo para um novo ficheiro <nome do ficheiro>.substituir, substituindo cada ocorrência de s1 por s2.

A utilização de funções de manipulação de ficheiros C é proibida e será considerada batota. Todas as funções de membro da classe std::string são permitidas, excepto a substituição. Utilize-as sabiamente!

É claro, lidar com entradas e erros inesperados. Tem de criar e entregar os seus próprios testes para garantir que o seu programa funciona como esperado.

Capítulo VIII

Exercício 05: Harl 2.0

	Exercício : 05
Harl 2.0	
Directório de entrada : *ex05/*	
Ficheiros para entregar : Makefile, main.cpp, Harl.{h, hpp}, Harl.cpp	
Funções proibidas : Nenhuma	

Conhece Harl? Todos conhecemos, não é verdade? Caso não conheça, encontre abaixo o tipo de comentários que Harl faz. Eles são classificados por níveis:

- Nível "**DEBUG**": As mensagens de debug contêm informação contextual. São sobretudo utilizadas para o diagnóstico de problemas.
  Exemplo: *"Adoro ter bacon extra para o meu hambúrguer de ketchup 7XL-douple cheese-triple-pickle-special- ketchup. Adoro mesmo!"*

- Nível "**INFO**": Estas mensagens contêm informação extensiva. São úteis para rastrear a execução de programas num ambiente de produção.
  Exemplo: *"Não acredito que acrescentar bacon extra custa mais dinheiro. Não puseste toucinho suficiente no meu hambúrguer! Se o fizesse, eu não estaria a pedir mais"!*

- Nível "**ADVERTÊNCIA**": As mensagens de advertência indicam um problema potencial no sistema. No entanto, pode ser tratado ou ignorado.
  Exemplo: *"Acho que mereço ter algum bacon extra de graça. Já venho há anos, enquanto que o senhor começou a trabalhar aqui desde o mês passado".*

- Nível "**ERROR**": Estas mensagens indicam a ocorrência de um erro irrecuperável. Esta é normalmente uma questão crítica que requer intervenção manual.
  Exemplo: *"Isto é inaceitável! Quero falar agora com o gerente".*

Vai automatizar Harl. Não será difícil uma vez que diz sempre as mesmas coisas. Tem de criar uma aula de **Harl** com as seguintes funções de membro privado:

- void debug(void);
- informação nula(nula);
- aviso de nulidade(nulidade);
- erro nulo(nulo);

Harl também tem uma função de membro público que chama as quatro funções de membro acima, dependendo do nível passado como parâmetro:

```
nulo      queixar-se( std::nível de string );
```

O objectivo deste exercício é utilizar **apontadores para as funções dos membros**. Isto não é uma sugestão. Harl tem de reclamar sem utilizar uma floresta de if/else if/else. Não pensa duas vezes!

Criar e entregar testes para mostrar que Harl reclama muito. Pode usar os comentários de exemplo.

Capítulo IX

Exercício 06: Filtro de Harl

	Exercício : 06
Filtro Harl	
Anuário de entrada : *ex06/*	
Ficheiros para entregar : Makefile, main.cpp, Harl.{h, hpp}, Harl.cpp	
Funções proibidas : Nenhuma	

Por vezes não se quer prestar atenção a tudo o que Harl diz. Implemente um sistema para filtrar o que Harl diz, dependendo dos níveis de registo que queira ouvir.

Criar um programa que tome como parâmetro um dos quatro níveis. Ele exibirá todas as mensagens a partir deste nível e acima. Por exemplo:

```
$> ./harlFilter
"ADVERTÊNCIA" [
ADVERTÊNCIA ]
Acho que mereço ter algum bacon extra de graça.
Há anos que venho, enquanto que o senhor começou a trabalhar aqui desde o mês passado.

[ ERROR ]
Isto é inaceitável, quero falar agora com o gerente.

$> ./harlFilter "Não tenho a certeza do cansaço que estou
insignificantes."
```

Embora existam várias formas de lidar com Harl, uma das mais efectivas é a de o SWITCH.

Dê o nome de harlFilter ao seu executável.

Deve usar, e talvez descobrir, a declaração de troca neste exercício.

Pode passar este módulo sem fazer o exercício 06.